VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2014

Tomáš Vojkůvka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Vojkůvka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: CATHEDRAL Software, s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Gaura**

Konzultant bakalářské práce: Ing. Josef Šustr

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 5. května 2014 .....

Touto cestou bych rád poděkovat panu Ing. Josefu Šustrovi za věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce. Panu Ing. Janu Gaurovi za pomoc a cenné rady při zpracování této práce. V neposlední řadě, bych chtěl poděkovat všem, kteří mi jakkoliv pomohli, poradili čí mě jen povzbudili, při absolvování bakalářské praxe a přispěli tak k jejímu zdárnému dokončení.

Abstrakt

Bakalářská práce se zabývá popisem odborné praxe, kterou student vykonával ve firmě CATHEDRAL Software, s.r.o. V bakalářské práci autor uvádí odborný popis zaměření firmy a popis pracovního zařazení. Následně uvádí seznam jednotlivých zadaných úkolů, které jsou rozepsány, spolu s jejich řešením. Konec bakalářské práce je věnován získaným znalostem a dovednostem v průběhu studia na VŠ a byly uplatněny při řešení zadaných úkolů. Následuje celé zhodnocení absolvované praxe a dosažených výsledků.

Klíčová slova: bakalářská práce, odborná praxe, VŠ, LaTeX, Python, PDF, XML, XSL, HTML, ReportLab, Eclipse, Notepad++

Abstract

Bachelor thesis describes professional practice which student performed in company CATHEDRAL Software, s.r.o. In the thesis the author presents technical description of the focus of the company and job description. Subsequently lists the various assigned tasks, which are described, together with their solutions. The end of the bachelor thesis is devoted to the knowledge and skills acquired during their studies at university and have been applied in solving assigned tasks. The following is the entire assess pedagogical practice and the results achieved.

Keywords: bachelor thesis, professional practice, University, LaTeX, Python, PDF, XML, XSL, HTML, ReportLab, Eclipse, Notepad++

Seznam použitých zkratek a symbolů

PDF	–	Portable Document Format
XML	–	eXtensible Markup Language, značkovací jazyk
XSL	–	eXtensible Stylesheet Language
XSLT	–	XSL Transformations
XSL FO	–	XSL Formatting Objects
XPath	–	XML Path Language
HTML	–	HyperText Markup Language, značkovací jazyk pro hypertext
IS	–	Informační systém
ArisCAT	–	IS společnosti CATHEDRAL Software, s.r.o.

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Bakalářská práce je zaměřena na absolvování odborné praxe ve firmě, za účelem získání nových zkušeností. Pro svou praxi jsem si vybral společnost CATHEDRAL Software, s.r.o., která sídlí v městě Prostějov. Práce obsahuje stručný popis historie firmy a její odborné zaměření.

Cílem bakalářské praxe je vyzkoušet si práci ve firmě a využít tak nabytých znalosti získaných během studia. Celá praxe se zabývá programováním v jazyce Python. Hlavním úkolem je vytvořit program, který bude sloužit pro převod dat z databáze do podoby PDF dokumentu. Ten bude následně možno vytisknou, popřípadě uložit. Další úkol, kterým se na praxi se student zabývá je tvorba grafů. Data, která jsou rovněž uložena v databázi, se pomocí programu načtou a převedou do grafu. Následuje export do formátu PDF nebo zobrazení na webu.

V další části bakalářské práce se student věnuje popsání získaným znalostem, zkušenostem a dovednostem, které získal během svého tříletého studia na VŠ. Zároveň je využil při absolvování odborné praxe.

V neposlední řadě práce zahrnuje znalosti nově získané na odborné praxi, kterých student nabyl řešením zadaných problému a při vypracovávání jejich řešení. Tyto znalosti a zkušenosti, budou s jistotou využity při dalším studiu nebo v budoucím zaměstnání. Závěr práce se zabývá celkovým zhodnocením dosažených úspěšných či neúspěšných výsledků a průběhem absolvované praxe.

2 Zasazení praxe

2.1 Popis odborného zaměření firmy CATHEDRAL Software, s.r.o.

Historie

Společnost vznikla v polovině roku 1996 jako sdružení fyzických osob Cathedral Software & Hardware. V letech 1997 - 1998 firma rozvinula obchodní aktivity a začala dodávat, instalovat a spravovat informační systémy, návrhem, realizací a výstavbou počítačových a privátních sítí. Jedny z prvních IS vytvořené touto firmou byly IS Žaluzie, Hokej 2000 a IS pro výchovné ústavy.

Největší zlom v oblasti informačních systémů byl IS ArisCAT, kterým se společnost zabývá již od roku 2001. Systém byl vytvořen na základě dlouholetých získaných zkušeností a je optimalizován na zpracování obtížně konfigurovatelných výrobků. Dalším ze systémů, kterým se společnost zabývá, je IS Helios Orange, sloužící ke zpracování obchodu a skladové hospodářství.

Odborné zaměření

V současné době se společnost CATHEDRAL Software, s.r.o. zabývá poskytováním komplexních služeb v oblasti informačních a komunikačních technologiích. V první řadě vývojem informačních systémů, především se jedná o vývoj a dodávku modulárního IS ArisCAT. Mezi další aktivity firmy patří distribuce IS Helios Orange.

Dalším polem působnosti, kde můžeme zařadit zakázkovou tvorbu software podle přání a zadání zákazníků s případnými pozdějšími úpravami. Můžeme zde také zařadit tvorbu mobilních aplikací pro Android a Windows Phone. Dle požadavků se firma zaměřuje také na hardware, přesněji správa serverů a počítačových sítí.

Ke všem poskytovaným produktům a službám, které firma nabízí, zajišťuje také veškerá potřebná školení, konzultace a případná poradenství.

2.2 Popis pracovního zařazení

Pro praxi jsem si vybral firmu CATHEDRAL Software, s.r.o., ve které jsem následně absolvoval přijímací pohovor. V několika prvních týdnech probíhala školení věnovaná popisu chodu a zaměření firmy, problematice a projektům, na kterých v současnosti firma pracuje. Jedno ze školení se zabývalo projektem, do kterého mě zasvětil předchozí praktikant. Na tomto projektu jsem si vyzkoušel drobné práce, jako například úprava stylování atp.

Po zapracování jsem se zařadil do týmu programátorů, kteří vyvíjejí drobné webové aplikace (programy). Tyto aplikace jsou částmi IS ArisCAT. Aktuálně jsou vybrané části IS napsány v jazyku PHP. Našim úkolem bylo přeprogramovat jednotlivé části do jazyku Pythonu.

3 Zadané úkoly v průběhu praxe

3.1 Studie: Možnosti a způsoby generování grafů a pdf souborů

První zadaný úkol, který jsem na praxi dostal, bylo provést studii zabývající se nalezením možností a způsobů, jakými lze generovat grafy a dokumenty do PDF souborů.

3.2 Generování PDF dokumentů

Dalším úkolem bylo vytvoření programu pro generování PDF dokumentů se vstupními daty uloženými v XML souboru.

3.3 Generování grafů

Posledním zadaným úkolem bylo vytvořit program, jako v předchozím úkolu, který bude načítat XML soubor s daty. Na výstupu bude PDF soubor s vytvořeným grafem.

4 Zvolený způsob řešení zadaných úkolů

4.1 Studie: Možnosti a způsoby generování grafů a pdf souborů

4.1.1 Zadání

Výsledkem studie by mělo být popis nalezených možností, způsobů jakými lze z XML souboru generovat grafy a PDF dokumenty. Popis srovnání jednotlivých metod, v čem spočívají jejich výhody a nevýhody. Jednou z podmínek bylo, aby případné řešení bylo zdarma a volně dostupné (např. freeware knihovny, software atd).

4.1.2 Řešení

4.1.2.1 PDF dokumenty

V dnešní době se na internetu nachází spousta možností a postupů, jak generovat soubory do jakéhokoliv formátu. Můžeme si vybrat z nepřeberného množství návodů a rad. Rozhodl jsem se, že použiji jednu z mnoha knihoven pro Python a to ReportLab Toolkit. O této knihovně jsem si přečetl dost informací, ať už na stránkách výrobce nebo na diskuzních fórech. Důvody, proč jsem si tuto knihovnu vybral jsou velice jednoznačné. Zaujala mne především svou údajnou rychlostí a jednoduchostí, kterou disponuje při generování PDF souborů. Dalším důvodem, proč jsem si knihovnu ReportLab vybral bylo, že obsahuje také knihovnu pro generování grafů, což bylo jedním dalších úkolů. Přišlo mi tedy vhodné, využit jednu rozsáhlou knihovnu s dobrými referencemi pro oba zadané úkoly.

Reportlab Toolkit

ReportLab je jméno knihovny pro přímé generování PDF dokumentů přímo v prostředí programovacího jazyka Python a zároveň název společnosti, která jej vytvořila [1].

Reportlab Toolkit je jeden z nejpopulárnějších a nejrozšířenějších nástrojů pro tvorbu PDF napsaný v Pythonu. Jedná se o knihovnu s open-source (otevřeným kódem) vyvíjená společnosti ReportLab. Knihovna je ke stažení zdarma, ale existuje také placená verze ReportLab PLUS, jejíž cena se pohybuje okolo 100 liber/měsíc [2].

Knihovna umožňuje rychlé a snadné vytváření dokumentů, automatické nebo daty řízené vytváření. ReportLab Toolik se v průběhu let vyvíjela v přímé reakci na reálných zpravodajských potřebách. Knihovna zahrnuje tří základní (hlavní) vrstvy:

- **grafické plátno** - umožňující „kreslit" PDF stránky a také zahrnuje mnoho speciálních funkcí (obrysy, odkazy a mnoho dalšího),
- **grafy a widgety** - speciální knihovna pro tvorbu datové grafiky zahrnující velké množství obchodních a finančních grafů,
- **PLATYPUS** - rozvržení stránky a topografie pomocí skriptů - vytváří dokumenty z komponent jako jsou nadpisy, odstavce, písmo, tabulky, obrázky a vektorová grafika [3].

Knihovna ReportLab vytváří dokumenty PDF založené **na grafických příkazech bez mezikroků**. Z tohoto důvodu je velmi rychlá a flexibilní (protože používáte kompletní programovací jazyk).

Příklady použití:

- dynamické vytváření dokumentů pro web,
- vytváření zpráv a publikace databáze,
- vestavitelný tiskový systém pro další aplikace, včetně 'jazyka zpráv', takže i uživatelé mohou upravovat své vlastní zprávy,
- 'vestavěný systém' pro vytváření komplexních dokumentů obsahujících grafy, tabulky a texty jako je manažerské a statistické zprávy, vědecké dokumentace a další,
- **převod XML souboru do PDF v jednom kroku** [4].

Jako další možné řešení jsem zvolil metodu převodu XML souboru do PDF dokumentu pomocí XSL stylů. Na tento nápad mne přivedl pan Ing. Šustr. Pokusím se částečně svými slovy vysvětlit postup vzniku PDF dokumentu spojením XML souboru, který obsahuje data a pomocí jazyka XSL.

XML

XML je obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Je zjednodušenou podobou staršího jazyka SGML (univerzální značkovací metajazyk, umožňuje definovat značkovací jazyky pro různé účely a různé typy dat). Zpracování XML je podporováno řadou nástrojů a programovacích jazyků. Jazyk je určen především pro výměnu dat mezi aplikacemi a pro publikování dokumentů, nezabývá se vzhledem. Prezentace dokumentu (vzhled) může být definována pomocí kaskádových stylů (CSS). Další možností zpracování je transformace pomocí XSLT, XSL FO a XPath.

Důvodem vzniku XML byla nekompatibila formátů dokumentů různých firem. Každá firma používá jiný informační a operační systém a pro každý formát bylo nutné mít náležitý software. XML je jednoduchý a je zpracovatelný v jakémkoli textovém editoru. Jednou z hlavních výhod je možnost volby kódování, základním kódování je UTF-8. Pokud nepíšeme dokument v UTF-8, musíme kódování určit v XML deklaraci [5].

DTD

Každý XML dokument může vyhovovat určitému typu dokumentu. Definice typu dokumentu (DTD) přitom říká, které elementy a atributy můžeme v XML dokumentu použít. Popisuje také vzájemné vztahy elementů. DTD je užitečný nástroj, kde jednou z výhod je, že nám umožní hlídat, zda mají naše dokumenty správnou strukturu. Druhou výhodou je, že při použití standardních DTD (např. HTML nebo DocBook) budeme mít k dispozici např. definice stylů vhodných pro formátování dokumentace. DTD se k XML dokumentu připojí pomocí deklarace:

<!DOCTYPE «*kořenový element*» SYSTEM ``«*URL*»''>

URL přitom udává adresu nebo jméno souboru, ve kterém je DTD uloženo a kořenový element je jméno elementu, ve kterém bude obsažen celý dokument (instance DTD).

DTD obsahuje deklarace čtyř typů: deklarace elementů, atributů, entit a notací. Jednotlivé deklarace se pak zapisují ve tvaru:

<! «*typ deklarace*» «*název*» «*deklarace*» >

U deklarace elementu napíšeme na místo typ deklarace *ELEMENT*. Název je název našeho vybraného elementu a za ním následuje obsah. Do deklarace elementu můžeme například definovat jaké další elementy má obsahovat a v jakém pořadí. Můžeme také zvolit počet, kolikrát se může jednotlivý element v XML dokumentu vyskytovat atp. Při deklaraci atributů napíšeme místo typu deklarace slovo *ATTLIST*. Nejobecnějším typem deklarace atributu je *CDATA*, který umožňuje jako jeho hodnotu zadat libovolný textový řetězec. Pro deklaraci entity použijeme typ *ENITITY %*. Slouží k deklaraci často se opakujícímu textu [6].

XSL

XSL vznikl jako univerzální stylový jazyk, který by měl nabízet funkčnost všech ostatních existujících stylových jazyků a dále je rozšířit. Samostatná syntaxe jazyka je samozřejmě založena na XML, takže pro zpracování stylu lze použít všechny nástroje, které umějí pracovat s XML.

Samotný standard XSL *Version 1.0 W3C Working Draft* rozděluje jazyk na dvě části. První část jazyka obsahuje nástroje pro *transformaci* XML dokumentů. Této části se říká **XSLT**. Druhá část standardu pak definuje *formátovací objekty* **XSL FO**. Ty slouží jako abstraktní popis vzhledu stránky a jejich jednotlivých komponent.

XSLT se dá použit pro transformaci XML dokumentu do XML dokumentu s jinou strukturou, případně do HTML nebo textového formátu. Samozřejmě, že lze XSLT použít i pro transformaci, jejímž výstupem je dokument skládající se z formátovacích objektů. Takový dokument pak může být pomocí speciálních programů zformátován a zobrazen či vytištěn [6].

XPath

XPath je počítačový jazyk, pomocí kterého lze adresovat části XML dokumentu. Pomocí tohoto jazyka lze z XML dokumentu vybírat jednotlivé elementy a pracovat s jejich hodnotami a atributy. XPath se používá v mnoha aplikacích XML, mezi nejvýznamnější patří využití v XSLT. Jazyk XPath je standardem vydaným organizací W3C [7].

V XPathu lze zapisovat jednoduché výrazy, které vybírají části XML dokumentu. XML dokument je přitom chápán jako stromová struktura, kde jsou jednotlivé elementy, atributy a texty chápány jako uzly [8].

Na ukázce z výpisu 1 si ukážeme příklad použití jazyka XPath. Kdybychom například chtěli vybrat *dodavatele*, použijeme XPath výraz `/faktura/dodavatel`. Lomítko přitom odděluje jednu úroveň ve stromu. Znamená to, že *dodavatel* musí být dítětem *faktury*.

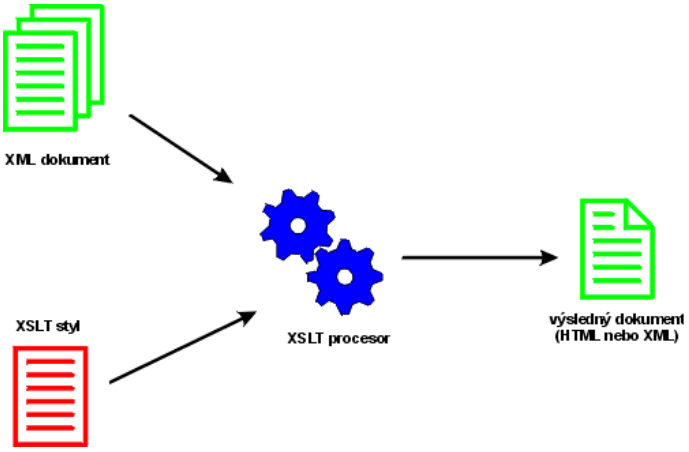
Pokud bychom chtěli odkázat na atribut faktury, například *datum_vystaveni* , napíšeme před jeho jméno `@` výraz bude vypadat `/faktura/@datum\_vystaveni`.

```
<?xml version="1.0" encoding="windows-1250"?>
<?xml-stylesheet href="faktura.xsl" type="text/xsl"?>
<!DOCTYPE vzorova_faktura SYSTEM "faktura.dtd">
<faktura cislo_faktury ="12345/022014" datum_vystaveni="2.2.2014" datum_splatnosti="3.3.2014">
  <web>www.web.cz</web>
  <dodavatel>
    <nazev>Corporation s.r.o.</nazev>
    <!-- dalsi elementy -->
  </dodavatel>
  <!-- dalsi elementy -->
</faktura>
```

Výpis 1: Ukázka XML kódu

Transformování XML dokumentu do PDF dokumentu

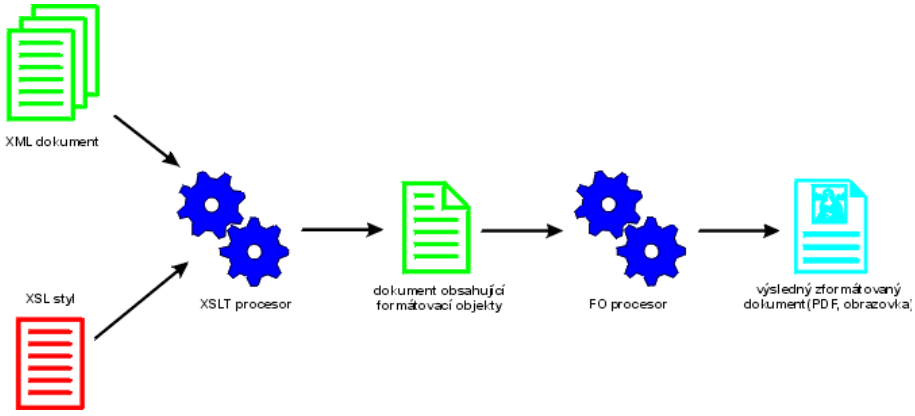
Víme, že soubory XML je možno pomocí jazyka XSL formátovat. Tento jazyk slouží k dvěma použitím, kdy první slouží k transformaci XML dokumentů do formátů HTML a používá se pro něj název XSLT. Druhé slouží k přesnému popisu vzhledu dokumentů a nazývá se XSL FO. K dispozici máme vše (jako například u CSS) včetně layotu (rozložení) stránky - sazba do sloupců, hlavičky, patičky apod.



Obrázek 1: Princip zpracování XML dokumentů pomocí stylů

Pokud chceme použít formátovací objekty pro zformátování našeho dokumentu, vytvoříme si tomu odpovídající styl. Tento styl pomocí XSLT aplikujeme na zdrojový XML dokument a dostaneme tedy XML dokument, který se bude skládat z jednotlivých formátovacích objektů.

Tento upravený dokument, který následně pomocí XSL FO zformátujeme (vyrenderujeme) do PDF souboru.



Obrázek 2: Formátování dokumentu pomocí XSL FO

Většinu procesorů lze využít buď jako samostatné programy nebo jako knihovnu vhodnou pro začlenění do vlastních programů.

V dokumentu (XSL) se objevují dva druhy značek - řídící příkazy pro procesor a značky výsledného dokumentu (např. HTML tagy). Ke kombinaci dvou sad značek se používají jmenné prostory. To znamená, že před jména všech elementů, které má XSLT procesor zpracovávat, se píše prefix `xsl:` celý styl musí být uzavřen v elementu `stylesheet`.

```
<xsl:stylesheet xmlns:xsl=http: // www.w3.org/1999/XSL/Transform version=”1.0”>
    // definice stylu
</xsl:stylesheet>
```

Výpis 2: Ukázka kódu pro styl

URL adresa `http://www.w3.org/1999/XSL/Transform`, která identifikuje jmenný prostor, je velmi důležitá. XSLT procesor podle ní pozná, že daný dokument opravdu styl obsahuje, pokud by tato URL v dokumentu nebyla uvedena nebo by obsahovala překlep, procesor instrukce ve stylu nebude vůbec zpracovávat.

Samotný styl se skládá ze šablon. Šablony definují, jak se mají jednotlivé části XML dokumentu převádět do tvaru výstupního dokumentu[9].

Šablony

Základem každého stylu jsou jsou šablony. Jejich základní tvar je:

```
<xsl:template match=”vyraz”>
    // telo sablony
</xsl:template>
```

Výpis 3: Ukázka kódu šablony

Tělo šablony přitom přesně definuje, jak se části transformovaného dokumentu vyhovující `vyrazu` budou zpracovávat. V těle šablony můžeme používat další konstrukce

XSLT nebo přímo elementy z výsledného dokumentu – nejčastěji tedy HTML tagy, protože generujeme HTML kód.

Mezi dva nejpožívanější příkazy, které se používají uvnitř šablony, patří *value-of* a *apply-templates*. Abychom pochopili k čemu se mají používat a jaký je mezi rozdíl, musíme nejprve vědět, v jakém pořadí se zpracovávají jednotlivé šablony.

XSLT procesor na začátku své práce načte do paměti vstupní XML dokument a vytvoří si jeho stromovou reprezentaci. Tento strom je pak postupně procházen od kořene v pořadí v jakém jsou elementy obsaženy v dokumentu (jedná se tedy o průchod do hloubky). V okamžiku, kdy je nalezena šablona odpovídajícího uzlu ve stromu, začne se její obsah zapisovat na výstup. Důležité je, že další potomci uzlu, pro který byla vybrána šablona, už nejsou dál automaticky zpracováváni. Pokud tak chceme učinit, musíme uvnitř šablony použít instrukci `<xsl:apply-templates/>`. Ta říká, že se má daná větev stromu zpracovávat dále a mají se pro její uzly hledat odpovídající šablony.

Pokud chceme v těle šablony použít jen textový obsah nějakého elementu a jeho podelementů, ale nechceme aplikovat další šablony, hodí se instrukce `<<xsl:value-of select="výraz"/>`. Ta vybere pouze obsah textových uzlů, které jsou potomky elementu určeného pomocí výrazu (ten je opět zapsán pomocí syntaxe XPath) [10].

Když se podíváme na oba způsoby, jakými lze generovat PDF dokumenty, jednoznačně můžeme říci, že metoda, kdy jsou data a styl rozděleny je mnohem lepší. Jediným problémem je, že nejdříve musíme vygenerovat dokument do formátu HTML a následně až do PDF dokumentu z výše zmiňovaných důvodů. Použití totiž nemusí sloužit pro generování pouze jednoho typu dokumentu, například faktury. Využít jej můžeme v podstatě pro jakýkoliv typ dokumentu si vybereme, stačí jen, když k příslušnému XML dokumentu s daty budeme mít XSL soubor se styly, který nám bude data formátovat do potřebného vzhledu. Tímto způsobem můžeme samozřejmě vytvořit také faktury nepřeberných vzhledů, podle potřeb a přání zákazníka.

V případě použití knihovny ReportLab je použití trochu horší a přizpůsobování vzhledu pro jiné dokumenty by bylo dosti problematické a hlavně velmi nákladné.

4.1.2.1 Grafy

Jedním z možných řešení pro generování grafů, je použití knihovny ReportLab, kterou popisuji výše. Další možnost, kterou jsem při generování grafů využil bylo Google Charts a RGraph, které využívají JavaScriptových knihoven pro generování libovolných grafů. Google Charts má také vytvořenou knihovnu pro jazyk Python, bohužel jsem k této knihovně nenašel dokumentaci, proto jsem od této možnosti upustil a nebudu ji dále zmiňovat.

RGraph

RGraph je knihovna, která je vytvořená pro tvorbu webových grafů. Pomocí JavaScriptu a HTML5 elementu `<canvas>` umožňuje vytváření nejrůznějších grafů uvnitř webového prohlížeče. Hlavními výhodami je rychlejší načítání stránek a menší zátěž webového serveru[11].

Knihovna je zdarma volně ke stažení a je možno jí použít a upravovat, dle potřeb. Na stránkách vývojáře můžeme nalézt kompletní dokumentaci s ukázkou použití.

4.2 Generování PDF dokumentů

4.2.1 Zadání

Program napsaný v jazyce Python bude načítat data uložená v XML souboru, která se budou získávat ze vzdáleného serveru. Následně budou naformátována, převedena a uložena do podoby PDF dokumentu. Výsledný dokument bude možno následně zobrazit nebo vytisknout. Prioritní využití bude na generování objednávkových faktur, které se budou lišit jak svým obsahem tak i stylem, podle toho jak si zákazník zvolí.

4.2.2 Řešení

V prvé řadě, před začátkem programování bylo potřeba vytvořit testovací XML soubor s daty, který bude následně sloužit pro generování PDF dokumentů. Jelikož jsem z XML z předchozích let neměl nijak velké zkušenosti, obstaral jsem si knihu s názvem *XML pro každého* od pana Jiřího Koska. Tato kniha se zabývá podrobným popisem tvorby XML a všech jeho součástí. Je možno si ji zakoupit nebo volně stáhnout na webových stránkách pana Koska. Díky této knize bylo snadné vytvořit XML soubor, který reprezentoval data faktury. Soubor přikládám v příloze pod názvem faktura.xml.

Jedním z prvních kroků bylo nainstalovaní jazyka Python. Pro programování, po domluvě s panem Ing. Šustrem, jsme zvolili vývojové prostředí Eclipse.[1] Abychom mohli v prostředí Eclipsu programovat v jazyce Python, bylo zapotřebí doinstalovat plugin PyDev.[2] Dále bylo nutno stáhnou a doinstalovat knihovnu ReportLab, kterou zmiňuji výše.

Vytvořený vzhled výsledné faktury (dokumentu) jsem zvolil podle svého uvážení. Inspirací, jak by měly prvky na faktuře být umístěny a co všechno by faktura měla obsahovat, jsem čerpal z faktur, které jsem nalezl doma.

Při tvorbě programu jsem začal nejdříve s pár základními importy, jako je velikost stránky, odstavce, tabulky a styly tabulek. Při postupném tvoření jsem přidával další importy např. pro barvu, rozměry, posléze i pro vytvoření „vlastního" písma.

Celý program je napsán v jedné třídě s názvem `PDFcreator`. Tato třída obsahuje následně všechny jednotlivé funkce programu. Jednou z prvních funkcí je `generatePDFfile`. Funkce obsahuje nastavení výsledného PDF dokumentu, jako je např. velikost stránky, styly písma a také načtený XML soubor, s jehož daty můžeme následně pracovat. Obsahuje také všechny vykreslované prvky (text, tabulky, atp).

První věcí, která se do PDF dokumentu vykresluje je hlavička. Hlavička obsahuje web firmy, která fakturu vystavila. Další je název, který určuje o jaký druh dokumentu se jedná,

[1] Eclipse je open source vývojové prostředí určené pro programování v jazyce Java. Pluginy umožňují toto vývojové prostředí rozšířit například o návrh UML, či zápis HTML nebo XML.

[2] PyDev je plugin třetí strany pro Eclipe. Je to integrované vývojové prostředí, pro programování v jazyce Python. Podporuje refactoring, grafické ladění, analýza kódu atp.

v mém případě jsem zvolil "Faktura - daňový doklad". Jako poslední prvek hlavičky je číslo faktury, jež je dle mého nejdůležitější a při hledání bude jistě užitečné jeho umístění v pravém horním rohu dokumentu. Všechny tři prvky jsou umístěny v jednom řádku, kdy web firmy je zarovnán k levému okraji, název dokumentu je centrován na střed a číslo faktury je zarovnáváno k pravému okraji dokumentu. Pro umístění do řádku a lepší pozicování, jsem zvolil umístění do tabulky. Tabulce je nastavena absolutní pozice, na kterém místě se má zobrazovat. Načtená data jsou uložena do předem vytvořených listů, kde jejich obsah je následně umístěn do tabulek. Vše se následně uloží do předem vytvořeného plátna a vykreslí.

Jako další se zobrazuji informace o dodavateli a příjemci. Informace o dodavateli obsahují název, ulici, město, zemi, IČO, DIČ, telefon, email, číslo účtu, variabilní symbol a způsob platby. U odběratele je informací o něco méně, postačí nám jméno a příjmení, město, ulice a země. V některých případech i telefonní číslo. Program obsahuje funkci s názvem `getXMLObject`, ve zkráceném popisu má tato funkce za úkol otevřit XML soubor a vrátí nám rozparsovaný XML soubor. Díky rozparsování docílíme toho, že můžeme použít libovolný element nebo atribut našeho XML dokumentu bez ohledu na to, kde se nachází. Při výpisu jednotlivých částí faktury jsem této možnosti využil, jako např. u dodavatele. Do buňky tabulky umisťuji všechny informace o dodavateli kromě čísla účtu, variabilního symbolu a způsobu platby. Pro příklad uvádím pouze fragment kódu, na které si ukážeme jak se data ukládají do stringu.

```
dodavatel = """Dodavatel<br/>
<br/>
\%s<br/>
 ...
Email:␣\%s""" \% (xml.dodavatel.nazev, ... , xml.dodavatel.email)
```

Výpis 4: Část kódu pro ukládání dat z XML souboru do stringu

Když nahlédneme do příloh na vzorovou fakturu, uvidíme, že dodavatel je dítětem faktury a název je dítětem dodavatele. Proto při použití tohoto fragmentu kódu xml.dodavatel.nazev se nám načte název a následně se vloží místo (za) první značku %s. Stejný „proces" proběhne u všech elementů dodavatele. Email je posledním vypsaným elementem spolu s tím, že před vypsaný email se nám vloží text "Email: ". Následně se string dodavatel vloží jako jeden parametr odstavce `Paragraph`, který se vloží do tabulky. Následuje stejný způsob uložení do plátna a vykreslení jako u hlavičky. Díky *Paragraphu* můžeme text upravovat pomocí HTML tagů, jak můžeme vidět ve výpisu 4. Kdybychom chtěli použít tučné písmo docílili bychom toho například takto:

```
<b>Email: \%s</b>
```

Tímto způsobem jsou vykreslovány všechny prvky zobrazené ve výsledném vygenerovaném PDF dokumentu. S trochu odlišným způsobem jsou načítány položky. Jelikož položek může obsahovat objednávka vždy více a přesný počet není předem znám, slouží pro načítání cyklus for, jež prochází XML soubor a hledá element s názvem `polozky`, ve kterém jsou jíž umístěny jednotlivé položky faktury zvlášť.

Při vykreslování jednotlivých prvků docházelo k tomu, že při zadání pozice pro zobrazení prvku (souřadnice) se prvky zobrazovaly na spodku vygenerovaného PDF dokumentu. Nedokáži přesně popsat, čím byl tento problém způsoben. Proto jsem hledal odpověď pomocí internetu, kde jsem jí následně našel na fóru `www.stackoverflow.com`. Řešili zde stejný problém překrývání prvků v dokumentu a byla zde také funkce, která zajišťuje to, že se prvky vykreslí opravdu na tom místě, které jím pomocí parametrů určíme.

Některými částmi kódu jsem se inspiroval na různých internetových fórech a stránkách. Například, když jsem potřeboval zarovnání tabulek nebo udělat v tabulce horizontální čáru atp.

Program je jakousi první verzi, pokusem, kterým jsem začínal. Při konzultaci s panem Ing. Šustrem jsme dospěli k závěru, že tato možnost je bohužel pro jejich potřeby nepoužitelná a nevyhovující. Při potřebě změnit vzhled výsledného PDF dokumentu by bylo velmi nákladné a náročné přepisovat celý kód programu. Z tohoto důvodu je program v rozpracované verzi a není nijak dokončen, např. při vygenerování PDF dokumentu se místo českých znaků zobrazí černé čtverečky.

Z výše jíž zmíněných důvodů jsem byl nucen začít pracovat na nové verzi programu. Základním principem je „procesor", který data uložená v XML souboru spojí se (aplikuje) styly uložené v XSL souboru a vygeneruje HTML nebo XML soubor, který je následně převeden do PDF dokumentu. Podrobnější postup a způsob převodu, jakým se PDF dokument tvoří je popsán ve studii výše. Program funguje na zhruba stejném principu. Jazyk XSLT jsem v tomto případě nepoužil, protože XSLT procesor, který slouží pro převod XML dokumentu do PDF, je napsán v jazyce Java. Určitě by bylo možno jej externě volat. Já jsem zvolil možnost převodu nejdříve do HTML souboru a následně do PDF.

Bylo zapotřebí vytvořit si soubor XSL, který obsahuje styly. Jako v předchozím případě, jsem zvolil stejný vzhled výsledné faktury. Při psaní stylu jsem použil jazyk CSS. Kostra layout dokumentu je tvořena pomocí tabulek v HTML. Styl se na XML soubor aplikuje tak, že připojíme stylový soubor pomocí příkazu:

```
<?xml-stylesheet href="faktura.xsl" type="text/xsl"?>
```

Který obsahuje název XSL souboru a typ daného souboru, v našem případě bude typ "text/xsl". Pokud bychom chtěli připojit pouze soubor s CSS styly byla by hodnota `type` "text/css". K souboru XML je dále připojen DTD soubor, který určuje typ dokumentu a hlídá nám, aby měl soubor správnou strukturu. U této příležitosti jsem upravil XML soubor pro větší přehlednost a přidal doplňující atributy.

Na začátku programu se načtou oba soubory, jak XML soubor tak i XSL soubor. XSL soubor se rozparsuje pomocí knihovny libxml2. V dalším kroku se rozparsuje XML soubor, na který se následně aplikuje XSL styl. Nakonec se vytvoří HTML soubor, který obsahuje data z XML souboru a vzhled z XSL souboru. Jako poslední následuje uvolnění (dealokace) jednotlivých souborů. Tímto kódem jsem se inspiroval na internetu a je drobně upraven. Před začátkem načítání souborů se zaznamená aktuální čas. Stejný krok se

provede následně na konci. Výsledné časy se od sebe odečtou a my dostaneme přehled o tom, jak dlouho program probíhal.

V dalším kroku měla následovat část, kdy se vytvořený HTML soubor převede pomocí knihovny do PDF dokumentu. Bohužel nastaly komplikace s použitou knihovnou s názvem pdfkit, která využívá knihovnu wkhtmltopdf. Po konzultaci problému s panem Ing. Gaurou, který mi poradil, abych umístil cestu do systémové proměnné %PATH%. Nyní při zadání příkazu do příkazové řádky např.:

```
wkhtmltopdf http://google.com google.pdf
```

Vše proběhne v pořádku a knihovna vygeneruje PDF soubor s obsahem stránky `www.google.com`. Zkoušel jsem tento příklad i na lokálním souboru a také vše fungovalo bez problémů. V programu Eclipse se mi knihovna nepodařila zprovoznit.

4.2.3 Zhodnocení, závěr

Výstupem programu je soubor HTML. Převod do dokumentu PDF je řešen pomocí funkce Tisk v prohlížeči. Pomocí této funkce můžeme dokumenty vytisknou nebo uložit. Možnost převodu z HTML do PDF pomocí jazyku Python existuje, ale v mém případě se ji nepodařilo zprovoznit. Její výhodou je hlavně nezávislost na vstupních datech. Pokud k patřičnému XML souboru máme napsaný i patřičný XSL dokument se styly, lze ji využít k vytvoření jakéhokoliv dokumentu.

4.3 Generování grafů

4.3.1 Zadání

Program pro generování grafů, bude jak v předchozím úkolu načítat data uložená v XML souboru. Tyto data budou načtena a následně z nich bude vytvořen graf, který se uloží do PDF dokumentu, případně bude možnost si výsledná graf zobrazit v prohlížeči.

4.3.2 Řešení

Řešení tohoto úkolu bylo o něco složitější než úkol předchozí, jelikož jsem s generováním grafů v jakémkoliv jazyce neměl žádné zkušenosti. Jako první možnost jsem zvolil knihovnu ReportLab. Na stránkách vývojářů je k nahlédnutí několik vzorových grafů.

Pro začátek jsem zvolil barový graf, který nenačítal hodnoty z XML souboru, jak je napsáno v zadání, ale pouze zobrazoval defaultně zadané hodnoty.

Graf implementuje dvě knihovny. Jedna slouží v vytvoření "plátna" a druhá slouží k importu příslušného námi zvoleného grafu. Zvolil jsem vertikální barový graf *VerticalBarChart*, což je graf který se vykresluje do sloupců od hozirontální osy. Nejprve si vytvoříme plátno, na které později vložíme námi předdefinovaný graf. Vytvoříme si nový graf a pojmenujeme si jej, já jsem zvolil název *chart* a zavoláme metodu `VerticalBarChart()`. Následně si můžeme určit jak velký má graf být, nejlépe je volit hodnoty stejné nebo menší než plátno, do kterého následně budeme graf vkládat. Potom je potřeba zvolit parametry

x a y, ty nám určují jakou vzdálenost od levého spodního rohu má graf mít, aby hodnoty na ose nezmizely za okraj. Dalšími parametry, které do grafu zadáváme jsou již samotné hodnoty zobrazované v grafu, následované popiskami. Posledním parametrem je nejmenší hodnota zobrazená na vertikální ose. Parametrů by šlo samozřejmě nastavit více např. legenda, barvy atp., ale já jsem pro začátek vybral jen ty nejdůležitější.

V posledním kroku se graf uloží do vytvořeného plátna a uloží se do zvoleného formátu. V mém případě možnost PDF dokument.

Po konzultaci s panem Ing. Šustrem jsem se rozhodl použít JavaScritovou knihovnu RGraph. Pro své použití jsem využil jeden z příkladů umístěných na stránkách vývojářů. Narozdíl od předchozího řešení jsem nyní použil lineární graf. Příklad jsem doplnil o jednoduchou funkci pro načítání dat z XML soubroru. Tuto funkci můžeme nalézt na stránkách `http://www.w3schools.com/dom/dom_parser.asp`, kde je volně k použití. Pro načítání hodnot jsem použil jednoduchý `for` cyklus, který prochází celý XML soubor a hledá hodnoty podle klíčových slov.

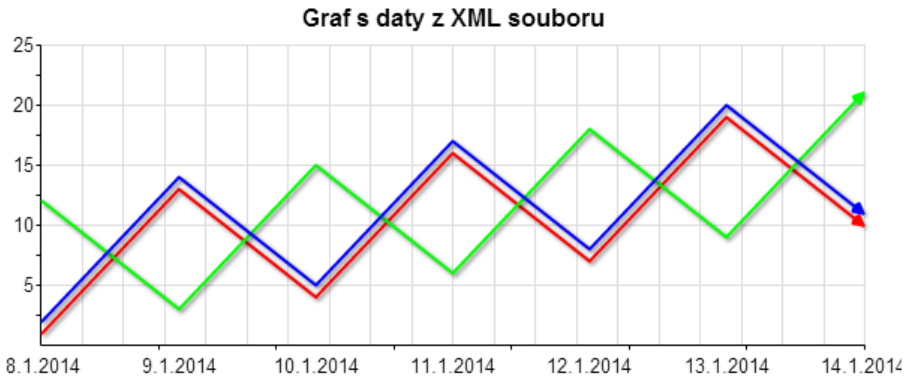
Když jsem se pokoušel graf spustit v jakémkoliv prohlížeči, všechny vypisovaly chybu, informující o tom, že není možné načíst soubor XML. V první chvíli jsem myslel, že problém je v onom souboru a proto jsem hledal chybu tam. Posléze se ukázalo, jak jsem se dočetl na internetu, že všechny dnešní moderní prohlížeče mají bezpečnostní opatření, aby nešlo načtení jiných souborů, které by mohly obsahovat nějaký skript stahující data apod. Řešením tohoto problému byla instalace webového serveru Apache, na kterém již graf z externího XML souboru bez problému načítal data.

```
<stats>
   <day datum="1.1.2014">
      <john>15</john>
      <fred>14</fred>
      <lucy>18</lucy>
   </day>
   <day datum="2.1.2014">
      <john>9</john>
      //
   </day>
</stats>
```

Výpis 5: Část XML souboru s daty grafu

Na vertikální ose se zobrazovala stupnice hodnot, vynesených v grafu. Hodnoty na horizontální ose jsou také načítány z XML souboru. Jsou uloženy pod názvem atributu datum. Pro jejich načítání jsem připsal do cyklu for podmínku, která ukládá jednotlivé datumy do předem vytvořeného pole stejně jako hodnoty v grafu.

Následuje samotné vytvoření, nastavené a zobrazení grafu. U grafu nastavujeme parametry: název, tloušťka zobrazené čáry, odsazení začátku čáry od okraje, zakončeni čáry grafu, dále popisky horizontální osy a legenda. Jako u knihovny ReportLab i zde existuje spousta možností nastavení, zvolil jsem pouze základní. Do těla stránky se pak umístí plátno s nastavenými velikostmi, do kterého se následně graf vykreslí.



Obrázek 3: RGraph s daty z XML souboru

4.3.3 Zhodnocení, závěr

Výsledkem práce je graf uložený v HTML souboru, načítající data ze souboru XML. Graf zobrazuje sedm dnů, kde každý den obsahuje tři hodnoty. Tento graf je velice jednoduchý a nezahrnuje žádnou složitější funkcionalitu jako např. dynamickou změnu popisků vodorovné osy podle počtu zobrazených dní. Pokud zadáme větší počet dní než je 7, budou se tedy dny vypisovat vedle sebe a následně se začnou přepisovat. Musíme tedy ručně zvětšit velikost grafu. Pro uložení můžeme rovněž využít funkci Tisk v prohlížeči.

5 Získané v průběhu studia uplatněné v průběhu odborné praxe

Při vykonávání odborné praxe, jsem využil mnoha znalostí a zkušeností získaných při studiu na VŠ, ale také mimo něj. Praxe vyžadovala především znalosti jazyka Python. Znalosti tohoto jazyka jsem získal v předmětech Skriptovací programovací jazyky a jejich aplikace (SPJA) a Uživatelská rozhraní (URO). Díky těmto znalostem pro mě bylo řešení zadaných úkolů o něco snazší, něž kdybych se musel jazyk učit od začátku. V hodinách těchto předmětů, především tedy SPJA, jsme se učili parsování XML souborů. Tyto znalosti jsem využil při generování PDF dokumentů. Programování v prostředí Eclipse mi také nebylo cizí, jelikož jsme v něm pracovali v předmětu Proramovací jazyk I (PJI).

Při tvorbě vzhledu PDF dokumentu jsem využil znalostí jazyka HTML a CSS, které jsem získal na střední škole při vypracovávání maturitní práce.

6 Znalosti a dovednosti získané v průběhu odborné praxe

Na praxi mi chyběla znalost jazyka XML a s ním spojených jazyků jako XSL, XPath atp. Proto bylo zapotřebí nastudovat si o těchto jazycích a tvorbě XML souboru potřebnou literaturu. Dále jsem také prohloubil své znalosti jazyka Python. U generování grafů jsem si osvojil jazyk JavaScript. Díky praxi jsem získal nové zkušenosti spojené s fungováním a chodem firmy.

7 Závěr

Hlavním cílem, proč jsem si tuto formu bakalářské práce zvolil, bylo, že mi přijde pro studenta, který studuje technicky zaměřený obor, co se týče získaných znalostí a zkušenosti, které se mi budou hodit ať už v budoucím zaměstnání nebo při dalším studiu lepší, než bakalářská práce klasickou formou.

U bakalářské praxe jsem prohloubil své znalosti z programování v jazyce Python, naučil jsem se nový jazyk XML a s ním spojené jazyky. Vyzkoušel jsem si své znalosti, které jsem se naučil během studia na VŠB a použil je v praxi při řešení zadaných úkolů. Mezi další klady bych zařadil komunikaci s lidmi při konzultacích, kdy jsem popisoval postupné pokroky v řešených úkolech. Dalším přínosem bylo nahlédnutí do fungování skutečné firmy. Dosažené znalosti a zkušenosti se pokusím rozšiřovat a dále na ně navazovat.

S výběrem společnosti CATHEDRAL Software, s.r.o. jsem byl velice spokojen. Konzultant pan Ing. Josef Šustr a další zaměstnanci firmy, byli vždy přívětiví a velmi ochotní pomoci.

Odbornou praxi, kterou jsem zde absolvoval, považuji za velice kladnou a příjemnou zkušenost do budoucna, které si cením.

8 Reference

[1] ReportLab. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2014 [cit. 2014-04-24]. Dostupné z: http://cs.wikipedia.org/wiki/ReportLab

[2] ReportLab's Software Tools: ReportLab PLUS. *ReportLab - Content to PDF Solutions* [online]. 2014 [cit. 2014-04-24]. Dostupné z: http://www.reportlab.com/software/

[3] ReportLab Toolkit. *ReportLab - Content to PDF Solutions* [online]. 2014 [cit. 2014-04-24]. Dostupné z: http://www.reportlab.com/software/opensource/rl-toolkit/

[4] Package: python-reportlab. *Debian* [online]. 1997-2013 [cit. 2014-04-24]. Dostupné z: https://packages.debian.org/cs/wheezy/python-reportlab

[5] Extensible Markup Language. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2014 [cit. 2014-04-24]. Dostupné z: http://cs.wikipedia.org/wiki/Extensible_Markup_Language

[6] KOSEK, Jiří. *XML pro každého: podrobný průvodce.* 1. vyd. Praha: Grada, 2000, 163 s. Průvodce (Grada). ISBN 80-716-9860-1

[7] XPath. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2014 [cit. 2014-04-24]. Dostupné z: http://cs.wikipedia.org/wiki/XPath

[8] XPath: Jazyk XSL. KOSEK, Jiří. *Domovská stránka Jirky Koska - "Vše o WWW"* [online]. 2000-2001 [cit. 2014-04-24]. Dostupné z: http://www.kosek.cz/clanky/swn-xml/ar07s66.html

[9] Jazyk XSL: aneb jak snadno a rychle transformovat XML dokumenty. KOSEK, Jiří. *Domovská stránka Jirky Koska - "Vše O WWW"* [online]. 2000-2001 [cit. 2014-04-24]. Dostupné z: http://www.kosek.cz/clanky/swn-xml/xsl.html

[10] Šablony: Jazyk XSL. KOSEK, Jiří. *Domovská stránka Jirky Koska - "Vše o WWW"* [online]. 2000-2001 [cit. 2014-04-24]. Dostupné z: http://www.kosek.cz/clanky/swn-xml/ar07s67.html

[11] RGraph: HTML5 charts library. *RGraph: HTML5 charts library - Free and Open Source* [online]. 2014 [cit. 2014-04-27]. Dostupné z: http://www.rgraph.net/

A faktura.xml

```
<?xml version="1.0" encoding="windows-1250"?>
<?xml-stylesheet href="faktura.xsl" type="text/xsl"?>
<!DOCTYPE faktura SYSTEM "faktura.dtd">
<faktura cislo_faktury="12345/022014" datum_vystaveni="2.2.2014" datum_splatnosti="3.3.2014">
  <web>www.web.cz</web>
  <dodavatel>
    <nazev>Corporation s.r.o.</nazev>
    <ulice>Street 1</ulice>
    <mesto>Town 12345</mesto>
    <zeme>Country</zeme>
    <ico>12345678</ico>
    <dic>CO87654321</dic>
    <telefon>+420 556 123 456, +420 123 456 789</telefon>
    <email>info@corporation.co</email>
    <cislo_uctu>1234567/0123</cislo_uctu>
    <variabilni_symbol>123456789</variabilni_symbol>
    <zpusob_platby>pevodem</zpusob_platby>
  </dodavatel>
  <odberatel>
    <nazev>Name Surname</nazev>
    <ulice>WallStreet</ulice>
    <mesto>BigTown 2</mesto>
    <zeme>Country</zeme>
    <telefon>+420 987 654 321</telefon>
    <email>name.surname@email.co</email>
  </odberatel>
  <polozka>
    <popis> znav  bunda</popis>
    <mnozstvi>1</mnozstvi>
    <cena_bez_dph mena="CZK">1950,75</cena_bez_dph>
    <dph_procenta>20,00</dph_procenta>
    <dph mena="CZK">390,25</dph>
    <cena_s_dph mena="CZK">2341,00</cena_s_dph>
  </polozka>
  <polozka>
    <popis> sk  pota </popis>
    <mnozstvi>1</mnozstvi>
    <cena_bez_dph mena="CZK">82,50</cena_bez_dph>
    <dph_procenta>15,00</dph_procenta>
    <dph mena="CZK">16,50</dph>
    <cena_s_dph mena="CZK">99,00</cena_s_dph>
  </polozka>
  <soucty>
    <cena_bez_dph_celkem mena="CZK">2033,25</cena_bez_dph_celkem>
    <dph_celkem mena="CZK">406,75</dph_celkem>
    <zaokrouhleni mena="CZK">0,00</zaokrouhleni>
    <cena_celkem mena="CZK">2440,00</cena_celkem>
  </soucty>
  <informace>Faktura slou i jako  reklama  list .</informace>
</faktura>
```

Výpis 6: XML soubor s daty

B faktura.html

www.web.cz **Faktura - daňový doklad** **12345/022014**

Dodavatel:	*Odběratel:*
Corporation s.r.o. Street 1 Town 12345 Country IČO: 12345678 DIČ: CO87654321 Telefon: +420 556 123 456, +420 123 456 789 Email: info@corporation.co	**Name Surname** WallStreet BigTown 2 Country Telefon: +420 987 654 321 Email: name.surname@email.co

Číslo účtu: 1234567/0123
Variabilní symbol: 123456789
Způsob platby: převodem

Datum vystavení: 2.2.2014
Datum splatnosti: **3.3.2014**

Popis	Množství	Cena bez DPH	DPH %	DPH	Celkem s DPH
značková bunda	1	1950,75	20,00	390,25	2341,00 CZK
Česká pošta	1	82,50	15,00	16,50	99,00 CZK

Celkem bez DPH	2033,25 CZK
DPH	406,75 CZK
Zaokrouhlení	0,00 CZK
Cena celkem	2440,00 CZK

Faktura slouží i jako reklamační list.

C Příloha na CD/DVD

Obsah:

- faktura.xml
- faktura.xsl
- faktura.html